UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Implementación de un servicio web basado en una red neuronal convolucional

Luis Chirlaque Hernández

Tutor: José Enrique López Patiño

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2020-21

Valencia, 3 de julio de 2021

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada
REG.NO. ES-CV- 000030

Agradecimientos

En primer lugar, a mi tutor por darme la oportunidad de desarrollar este proyecto.

A mis compañeros de clase y de piso, por recorrer el sendero a mi lado y tenderme la mano siempre que lo he necesitado.

A José Miguel, por descubrirme el mundo del Deep Learning y guiarme a través de él.

Por último, a mi familia, por darme la fuerza y el apoyo que necesito para llegar a donde he llegado.

Resumen

El Deep Learning está mostrando un gran potencial en infinidad de aplicaciones revolucionando múltiples campos. Sin embargo, a pesar de su potencial, es de difícil manejo debido a que requiere de conocimientos técnicos para su uso. Por ello, resulta interesante su despliegue en servidores web.

En este trabajo se pretende desarrollar un servidor web mediante el entorno multiplataforma Node.js, que implemente una red neuronal ya existente. El servidor desarrollado permitirá al cliente utilizar la red para separar las pistas de audio de manera configurable y sencilla.

Se evaluarán los resultados en base a la eficiencia y limitaciones del entorno viendo que ventajas e inconvenientes ofrece y cómo ha resultado su despliegue.

Se explorará también la idea de expandirlo a otro tipo de redes neuronales.

PALABRAS CLAVE: red neuronal, servidor web, node.js, JavaScript, back end, Python, aprendizaje profundo

Abstract

Deep Learning is showing a huge potential in a big number of applications, revolutionizing multiple fields. However, in spite of its potential, it is hard handling it due to the technical knowledge it requires to be used. Thus, its deployment in a web server results interesting.

This assignment is pretended to develop a web server using the multiplatform environment Node.js to implement an already existing neural network. The developed server will allow the clients to use the neural network in order to separate the audio tracks in an easy and configurable way.

The results will be evaluated according to the efficiency and limitations of the environment, analyzing the advantages and disadvantages it offers and how the deployment has been achieved.

The idea of expanding it to other types of neural networks will also be explored.

KEYWORDS: neural network, web server, node.js, JavaScript, back end, Python, deep learning

Índice

Índice de Figuras

Capítulo 1. Introducción

Nuevas tecnologías van surgiendo para la implantación de servidores y servicios web. Muchas ofrecen novedades en la forma de desarrollar estos servicios y cada vez se van utilizando más en entornos profesionales.

De la misma manera, el Deep Learning se establece como una de las tecnologías dominantes dentro de la inteligencia artificial que está viviendo nuevamente un auge, convirtiéndose en una de las bases fundamentales de la transformación digital. Estos nuevos algoritmos permiten computar y automatizar nuevas tareas y datos que hasta ahora no se habían conseguido.

1.1 Motivación

Este trabajo ha supuesto para mí el culmen de dos de mis grandes pasiones: la música y la tecnología.

Ya suman más de 10 años los que llevo formándome en el mundo de la música, y es que es algo que a día de hoy me sigue apasionando. Gracias a mis estudios del Grado de Telecomunicaciones he podido comprender mucho mejor otros aspectos quizá algo más técnicos que están también directamente relacionados con ella.

Además, me he iniciado recientemente en el mundo de la inteligencia artificial. He encontrado en este campo algo todavía enigmático para mí, con un sinfín de posibilidades para resolver problemas complejos.

Por último, gracias a mis estudios en telecomunicaciones soy capaz de desarrollar proyectos personales de forma eficaz. Esta es la herramienta que me va a permitir unificar el mundo de la música con tecnologías de lo más novedosas.

Todo esto ha sido lo que me ha llevado a plantear un proyecto que unificase las cosas que más me apasionan y también ha sido el motor que me ha inspirado para poder llevarlo todo a cabo.

1.2 Objetivos

Este trabajo va a cubrir distintos objetivos, desde el nacimiento hasta la ejecución completa y testeo de la idea.

Por una parte, se pretende explorar la situación del Deep Learning enfocado a la separación de audio y el estudio de las distintas alternativas por las que una red de estas características pueda ser implementada como un servicio web.

El siguiente objetivo pretende enlazarse con las ideas del primero, dando vida a una implementación real con una de las tecnologías disponibles y la red seleccionada para hacer una prueba. Esto supondrá diseñar y desarrollar todo el funcionamiento de un servicio web tomando solo como punto de inicio la red de separación de pistas ya entrenada por sus desarrolladores. Este servicio permitirá utilizar la red seleccionando la pista de audio que queramos junto a algunos parámetros de configuración, siendo sencillo, accesible y fácil de utilizar para cualquier usuario.

Adicionalmente, se implementará una interfaz API para el uso no solo web sino también REST de este servicio web, permitiendo que otras aplicaciones puedan hacer uso de este servicio de forma automatizada. Además, se desplegará en un servidor privado virtual para simular la complejidad de un despliegue de estas características.

Por último, se comentarán las dificultades que surjan durante todo el proceso, se analizarán y valorarán los resultados comentando además algunas posibles mejoras. También se presentarán algunas ideas para el desarrollo futuro que puede seguir la aplicación.

1.3 Estructura del proyecto

El proyecto va a dividirse en varias fases. En primer lugar, una primera fase de desarrollo de la idea.

A continuación, una segunda fase en la que se estudiará todo lo necesario para poder abordar el proyecto, valorando diferentes alternativas tecnológicas.

Seguidamente, la fase de desarrollo de la aplicación. En ella, se pondrá en marcha todo lo aprendido anteriormente para poder desarrollar el servicio web. Esta será la parte más técnica pues abordará la programación del servicio.

Después, una fase de test para verificar el correcto funcionamiento y arreglar posibles fallos.

Luego, se desplegará el proyecto en un servidor virtual.

Figura 1. Estructura del proyecto

Finalmente se comentarán los problemas, evaluarán los resultados y se estudiará una propuesta de futuro.

Capítulo 2. Estado del arte

2.1 Procesamiento de señales de audio

El sonido se conforma por ondas mecánicas producidas por cambios de presión en el aire. Estas perturbaciones pueden ser recogidas por el diafragma de un micrófono, que convierte estos desplazamientos de presión en una serie de medidas en el tiempo, generando una forma de onda. Cuando reproducimos esta misma forma de onda en un altavoz, podemos recrear los mismos sonidos que fueron captados.

Las señales multicanal están formadas por varias formas de ondas capturadas por uno o más micrófonos y mezcladas. Normalmente, las señales de música son estereofónicas, lo que quiere decir que constan de dos formas de onda.

La frecuencia de muestreo de una señal es el número de muestras que se toman de la señal por cada unidad de tiempo para convertirlas en una señal analógica. En telefonía se suele utilizar un valor típico de 8kHz, que se considera que es suficiente para poder mantener una conversación fluida y reconocible. No obstante, es cierto que, para conseguir una calidad auditiva mayor, se ha de usar una frecuencia de muestreo mayor. Los valores típicos para esto los encontramos en 44.1kHz y 48kHz.

Para el procesado en el dominio de la frecuencia una herramienta fundamental es la Transformada de Fourier de Tiempo Reducido (STFT). Cuando utilizamos la Transformada de Fourier clásica para obtener el espectro musical de una canción nos encontramos con el problema de que obtenemos el espectro de todo el rango temporal de forma simultánea. Esta información por lo general no suele ser de mucha utilidad, por lo que se suele recurrir a una versión de esta en segmentos de tiempo mucho más reducidos, proporcionándonos información por unidad de tiempo. Esta transformada es una secuencia de transformadas de Fourier en ventanas de tiempo. Estos segmentos de tiempo se conocen como ventanas e influyen en la resolución temporal-frecuencial.

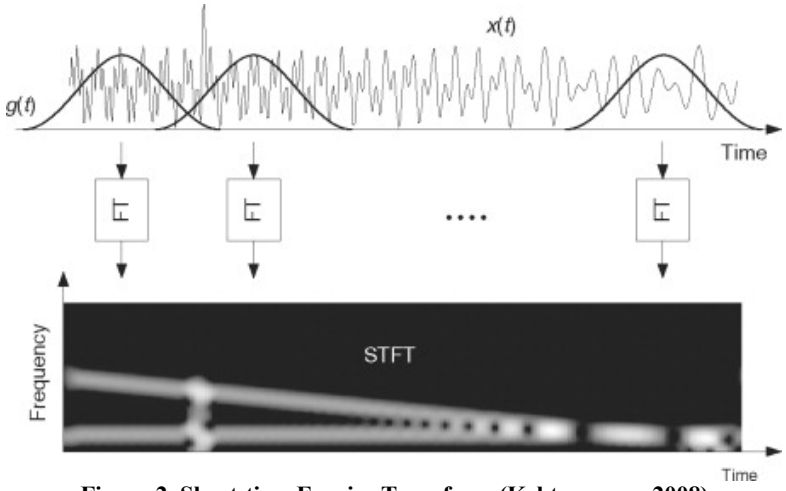

Figura 2. Short-time Fourier Transform (Kehtarnavaz, 2008)

En la Figura 2 se pueden apreciar las distintas ventanas y su representación espectral. Con esta representación, se pueden visualizar los cambios en frecuencia a lo largo del tiempo. Esto se conoce como espectrograma.

El problema de la separación de distintas fuentes sonoras lleva siendo un quebradero de cabeza para los investigadores dentro de la comunidad musical durante varias décadas. La grabación de música suele hacerse grabando diferentes pistas por separado (batería, guitarra, bajo…) y más tarde se mezclan todas en una sola pista. La tarea de separar las pistas es simplemente el proceso contrario. Dada una mezcla, conseguir recuperar las pistas de cada instrumento por separado.

La finalidad se puede encontrar en hacer nuevas mezclas, eliminar una pista para hacer una *cover* o una versión para karaoke, aislar una para poder enfocarse en esa pista con otros propósitos o podría servir para el clasificado automático de canciones en determinados géneros musicales.

El reto consiste en conseguir recuperar de la manera más fidedigna posible las distintas pistas que componen una canción, o al menos lo más aproximado posible, introduciendo la mínima dispersión posible.

Esta tarea podía ser muy costosa y requería unos conocimientos técnicos muy altos y un software especializado.

2.2 Deep Learning

Curiosamente, nuestro cerebro es muy bueno a la hora de separar sonidos. Basta con concentrarse en uno de los instrumentos que están sonando, podemos ir siguiendo su movimiento melódico distintivamente del resto.

El avance del machine learning en los últimos años nos ha acercado cada vez a mejores soluciones ante este problema.

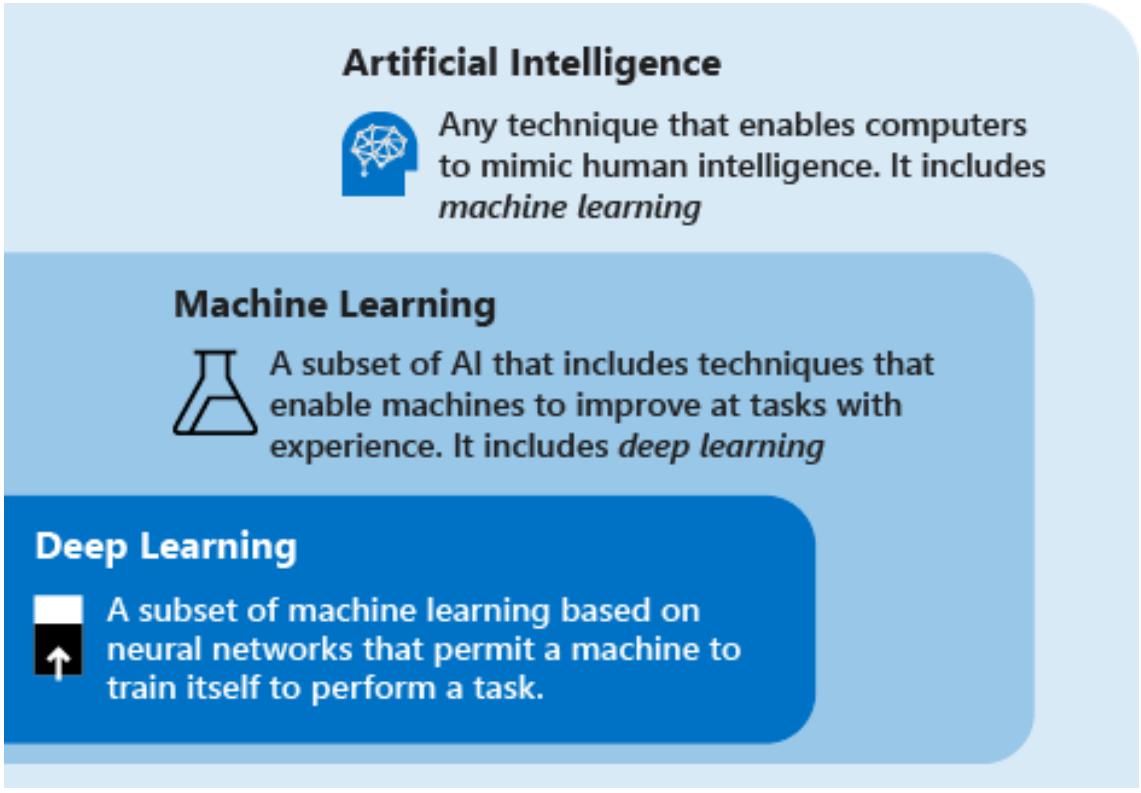

Figura 3. Aprendizaje profundo, Machine Learning e inteligencia artificial (Microsoft, 2021)

En la Figura 3 se ve una visión general de la inteligencia artificial, el aprendizaje automático y el aprendizaje profundo. Como se puede ver, la inteligencia artificial es el campo que engloba cualquier técnica con el objetivo de dotar a un ordenador con habilidades que imiten la inteligencia humana, el proceso del aprendizaje, el razonamiento y la propia corrección (Dobrev, 2005).

Dentro de este campo se encuentra el aprendizaje automático, que es un subconjunto de técnicas de IA capaces de aprender del entorno y de la experiencia (El Naqa & Murphy).

Por último, se encuentra dentro del aprendizaje automático al aprendizaje profundo. El aprendizaje profundo trata no solo de aprender la relación entre dos o más variables, sino también el conocimiento que gobierna las relaciones, así como el conocimiento que le da sentido (Zhang,

Yang, Lin, Ji, & Gupta, 2018). Este subconjunto utiliza métodos en cascada con múltiples capas de abstracción, formando una jerarquía de conceptos entre los diferentes niveles de abstracción.

El Deep Learning crece a una velocidad desmesurada ya que se requiere para el Data Science. Las empresas cada vez invierten más capital para poder procesar más y mejor los datos. Como ejemplo, podemos ver que hasta un 77% de los dispositivos que utilizamos diariamente hacen uso del Machine Learning (Dialani, 2020). Ya sea las recomendaciones que nos hace Netflix sobre que título ver a continuación, las ofertas que nos ofrece Amazon o el reconocimiento de voz cuando hablamos con Alexa.

Cada vez se está aplicando a nuevos campos, como la medicina, en la que se está explorando su uso para el reconocimiento de enfermedades mediante el análisis de imágenes, gracias especialmente a las redes neuronales convolucionales que son especialmente útiles en este tema.

Una característica fundamental necesaria para poder entrenar a este tipo de redes y que aprendan son los sets de datos. Se requieren sets de datos enormes para ir alimentando la red a la entrada y comparando los resultados obtenidos con los esperados y así poder 'enseñar' a la red.

2.3 Redes neuronales para la separación de audio

Uno de los tipos de redes más prometedoras hoy en día son las redes neuronales convolucionales. La diferencia con las redes neuronales convencionales es que redes convolucionales contienen múltiples capas con filtros convolucionales de una o varias dimensiones. El resto de funcionalidades como el aprendizaje o la extracción de características es muy semejante al de una red neuronal convencional.

Estos filtros convolucionales consisten en ir haciendo el producto escalar de los valores de la entrada con una matriz llamada *kernel*. Esto genera una nueva matriz de salida que contendrá la información extraída de la entrada inicial.

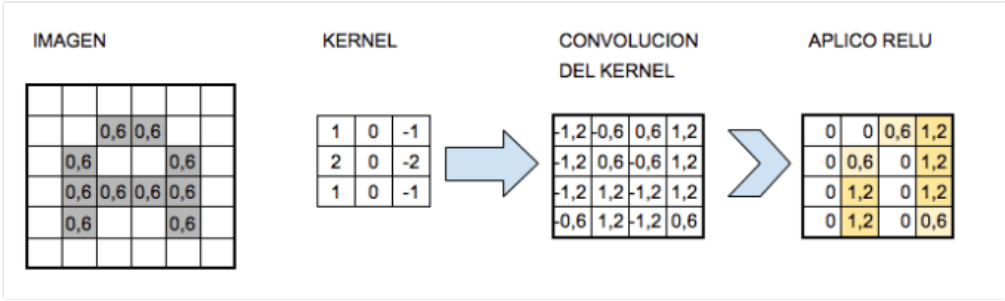

Figura 4. Filtro convolucional mediante kernel (Barrios, 2019)

En la Figura 4 se puede ver un ejemplo de un filtro convolucional aplicado a la extracción de características en imágenes, donde el kernel se va moviendo por todos los píxeles, generando una nueva matriz con los resultados de estas operaciones.

Para el correcto funcionamiento de estas redes se precisa de un entrenamiento previo. Muchas veces, lo más costoso es obtener un set de datos lo suficientemente grande y variado para que la red aprenda de forma adecuada. En el caso del audio, existen sets de datos como musdb18, DSD100 o MedleyDB entre otras que contienen cientos de canciones ya separadas. Gracias a la existencia de estos sets, muchas organizaciones han podido desarrollar sus propias redes de distinto tipo y con distintos enfoques para plantear cada vez un mejor funcionamiento en el problema de la separación de audio.

En la Tabla 1 podemos observar algunas de las redes más famosas que cumplen esta finalidad.

Model	Domain	Extra data?	Overall SDR	MOS Quality	MOS Contamination
Open-Unmix	spectrogram	no	5.3	3.0	3.3
D3Net	spectrogram	no	6.0	-	-
Wave-U-Net	waveform	no	3.2	-	-
Demucs	waveform	no	**6.3**	**3.2**	3.3
Conv-Tasnet (this)	waveform	no	5.7	2.9	**3.4**
Demucs	waveform	150 songs	**6.8**	-	-
Conv-Tasnet	waveform	150 songs	6.3	-	-
MMDenseLSTM	spectrogram	804 songs	6.0	-	-
D3Net	spectrogram	1.5k songs	6.7	-	-
Spleeter	spectrogram	25k songs	5.9	-	-

Tabla 1. Comparativa de precisión (Défossez, Usunier, Bottou, & Bach, 2021)

Pese a haber algunas otras redes con un SDR mayor como Demucs, se ha elegido Spleeter por estar desarrollada en Tensorflow y existir una buena relación con JavaScript, dada la existencia de Tensorflow.js. Otras como Demucs están desarrolladas con PyTorch y podría haber problemas de compatibilidades. Además, ambas redes se han probado antes y Demucs es computacionalmente más cara que Spleeter.

En concreto, Spleeter, desarrollado por el equipo de la compañía Deezer, cuenta con una arquitectura basada en redes 'U-Net'. Este tipo de redes tenía como propósito inicial la segmentación de imágenes para uso biomédico (Roinneberger, Fischer, & Brox, 2015) pero vemos como se ha expandido a otros usos completamente distintos.

No se va a entrar en detalle al funcionamiento de estas redes pues queda fuera del alcance de este trabajo.

Capítulo 3. Servicio web

3.1 Comparativa de tecnologías existentes

A la hora de desarrollar un servicio web existen multitud de tecnologías que se pueden adoptar para construir tal proyecto. Elegir una u otra puede suponer planear un proyecto de una manera totalmente distinta a otra. En este apartado se va a comparar algunas de las tecnologías más populares.

Entre las tecnologías existentes para el backend, las más populares son PHP, Node.js, Spring o Django.

3.1.1 PHP

PHP es un lenguaje de scripting de propósito general especialmente adecuado para el desarrollo web.

Ventajas:

- Fácil aprendizaje
- Gran cantidad de proyectos en este lenguaje
- Implementable prácticamente en cualquier servidor web
- Independiente de la plataforma
- Estable y robusto
- Open-source

Desventajas:

- Frameworks con curva de aprendizaje alta
- Control de errores débil
- Difícil de manejar en aplicaciones muy grandes al no ser modular.

3.1.2 Spring

Spring proporciona un modelo de programación para aplicaciones modernas basadas en Java, para cualquier tipo de plataforma de despliegue.

Ventajas:

- Ligero
- Modular
- Flexible
- Rápido
- Altamente configurable

Desventajas:

- Alta curva de aprendizaje
- XML, lo cual puede ser muy verboso

3.1.3 Django

Es un framework de Python pensado para el desarrollo rápido y limpio de webs.

Ventajas:

- Alta escalabilidad
- Procesamiento rápido
- Buena seguridad
- Utiliza Python

Desventajas:

- Alta curva de aprendizaje
- No es adecuado para proyectos pequeños
- Monolítico, las cosas deben ser hechas de determinada manera
- Framework muy pesado

3.1.4 Node.js

Es un entorno de tiempo de ejecución de JavaScript que incluye todo lo necesario para desarrollar aplicaciones web.

Ventajas:

- Gestor de paquetes excelente
- Alto rendimiento ante tareas ligeras
- Altamente escalable
- Utiliza Javascript
- Fácil de aprender
- Procesa peticiones por eventos
- Multiplataforma

Desventajas:

- Bajo rendimiento ante tareas computacionalmente pesadas

Se ha elegido trabajar con Node.js por trabajar con JavaScript, unificando los lenguajes del frontend y de backend. Además, es uno de los que menor curva de aprendizaje tiene. Su modelo de funcionamiento, que será explicado en el siguiente apartado es también muy llamativo para hacer aplicaciones de baja carga con muchos usuarios. Por último, existen algunas librerías para el aprendizaje profundo compatibles con este framework.

3.2 Descripción de la tecnología escogida

Node.js es un entorno en tiempo de ejecución multiplataforma y de código abierto de JavaScript. Esto permite dotar a JavaScript de nuevas funcionalidades con las que antes no se contaban y, por ende, se le puede dar otros usos que no sean solo la manipulación de páginas en el navegador. Por ejemplo, ejecutar código JavaScript en el lado del servidor y no sólo en el cliente. De esta manera, se puede construir una aplicación web con Node.js. Pero esta tecnología no sólo queda ahí. Puede abarcar otros campos ya que no es un lenguaje de programación, si no un entorno de ejecución para correr código JavaScript. Otros usos podrían ser scripts, build tools, programas…

Node.js utiliza el motor V8 de Google. Al ser multiplataforma, puede ser utilizado en cualquier máquina y solo hace falta visitar nodejs.org para descargar la última versión. Una vez se ha instalado, ya estará listo para ser usado.

El modo de funcionamiento característico de Node es su naturaleza asíncrona, no bloqueante y controlada por eventos. Es decir, otras tecnologías web tradicionales generan un subproceso cada vez que se establece una conexión con el servidor. Esto es limitante ya que se está reservando una cantidad de recursos (como memoria RAM) para distintos clientes que pueden no estar aprovechándolos. Esto también supone que pueda haber un límite de conexiones simultáneas. En cambio, Node.js funciona con un solo subproceso que va procesando todos los eventos que van entrando en el Event Loop. Cada vez que se genera una nueva solicitud, esta es introducida al bloque de eventos y una vez haya sido procesada se ejecutará un callback. Como ventaja principal, Node puede servir a muchos más clientes simultáneamente.

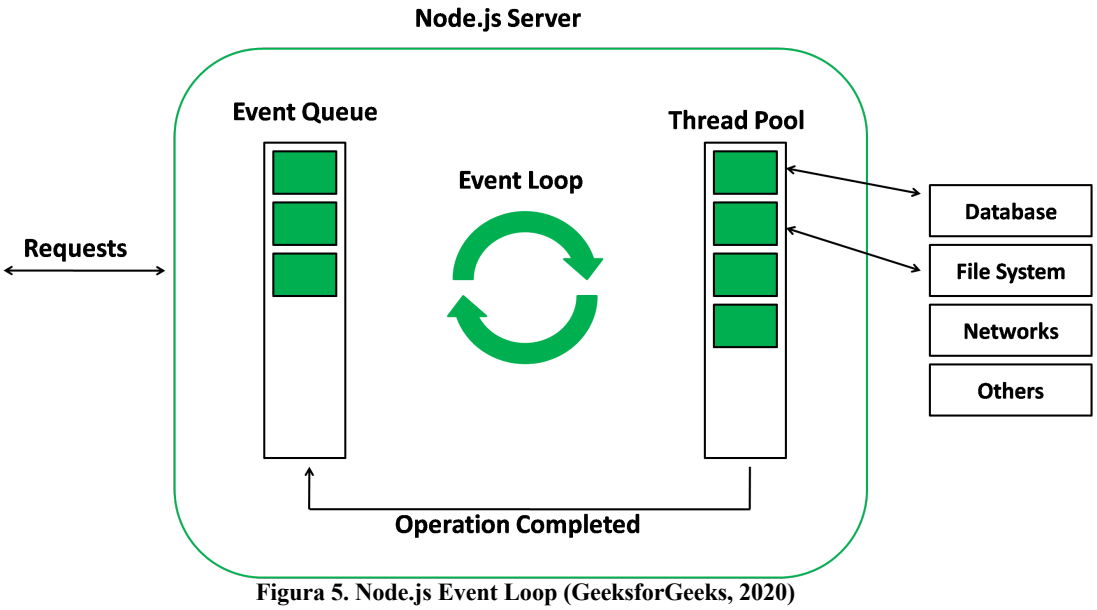

Figura 5. Node.js Event Loop (GeeksforGeeks, 2020)

En la Figura 5 podemos ver como el Event Loop va ejecutando todos los eventos de la Event Queue en un único hilo. Cuando se encuentra operaciones bloqueantes I/O, las introduce en el Thread Pool donde hay varios hilos concurrentes realizando únicamente este tipo de operaciones bloqueantes. Cuando han sido finalizadas la devuelven al Event Queue para su callback.

Este modelo cambia la forma en la que los desarrolladores han de manejar el proyecto. Al haber pocos hilos simultáneos, se han de usar de la forma más eficiente posible y evitar el bloqueo de cualquiera de los hilos. Si el Thread Pool recibe muchas tareas computacionalmente pesadas, la cola se verá ralentizada y el caudal del servidor sufrirá.

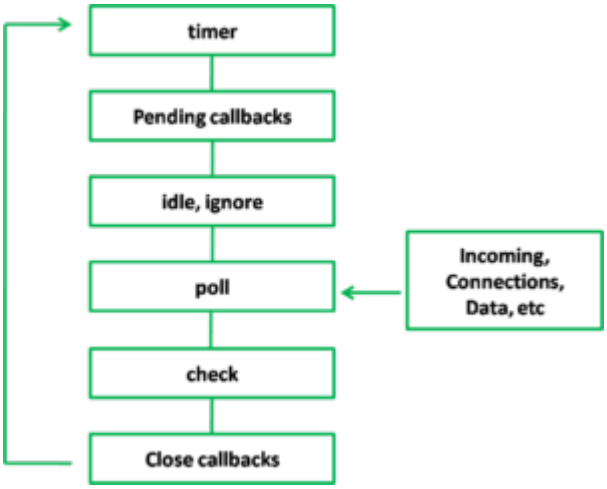

Figura 6. Fases del Event Loop (GeeksforGeeks, 2020)

En la Figura 6 se pueden observar las distintas fases que conforman el Event Loop. Cada una de estas fases tiene una cola FIFO de callbacks para ejecutar. Cuando el bucle entra en una fase, ejecuta todas las operaciones de dicha fase y posteriormente los callbacks hasta que la cola se haya vaciado por completo o se haya ejecutado el número máximo de callbacks. A continuación, el bucle avanza hasta la siguiente fase.

- Timers: Esta fase ejecuta los callbacks referentes a temporizadores, programados por setTimeout() y setInterval().
- Pending callbacks: Ejecuta los callbacks de E/S pospuestos a la siguiente iteración del bucle.

- Idle: Es usado internamente.
- Poll: Esta fase recupera los nuevos eventos de E/S. También ejecuta los callbacks relacionados con la E/S. Node puede adoptarse bloqueante durante esta fase.
- Check: Ejecuta los callbacks referentes a setImmediate().
- Close callbacks: Ejecuta las llamadas de cierre, como las de un socket o un buffer.

3.2.1 Framework

Existen numerosos frameworks que pueden ser utilizados con Node.js para el desarrollo de aplicaciones web, como por ejemplo Koa, Hapi o Sails. Para este servicio se ha decidido utilizar Express.js por ser uno de los más utilizados, tener una amplia comunidad y una buena documentación, siendo además uno de los más robustos.

Express.js es un framework de desarrollo de aplicaciones web minimalista y flexible para Node.js. Este framework va a ser el que se utilice para crear la aplicación web ya que va a simplificar y acelerar el desarrollo por su simplicidad y potencial. Va a permitir manejar las rutas de manera sencilla, utilizar motores de vista, agregar middlewares que serán explicados en el siguiente apartado y una fácil integración con la base de datos.

Una característica muy importante que se va a usar de Express es la clase Router. Esta clase permite simplificar el direccionamiento de las rutas, haciendo que estas puedan ser modulares.

3.2.2 Middleware

Middleware es una manera de dividir el programa en diferentes bloques por los cuales la petición va pasando y ejecutando esos trozos de código. La mayor ventaja que proporciona usar Middleware es que se puede dotar al programa de nuevas funcionalidades haciendo introduciendo nuevos middlewares, sin añadir complejidad al resto de funciones.

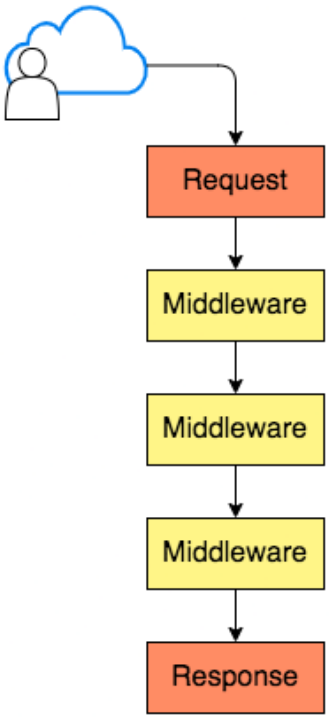

Figura 7. Funcionamiento del Middleware

El framework de Express.js está pensado para trabajar con Middleware y aporta múltiples facilidades para su integración. El método que permite añadir nuevas funciones de middleware en Express es use( );

Durante el proyecto se va a usar una gran cantidad de middlewares para dotar de distintas funcionalidades a modo de módulos a la aplicación.

3.3 Planificación del proyecto

3.3.1 Arquitectura

Para este proyecto se ha utilizado la arquitectura MVC (Modelo Vista Controlador).

Este patrón de desarrollo trata de estructurar un proyecto en tres componentes distintos:

- Modelo
- Vista
- Controlador

Con esta arquitectura cada componente tiene una tarea específica y permite adoptar una separación de conceptos importante a la hora de desarrollar la aplicación.

Los **modelos** son objetos que se encargan de representación de los datos de nuestro proyecto, permitiendo trabajar con ellos. Por ejemplo, insertar nuevos datos, modificarlos, buscar…

Las **vistas** se encargan de la parte visual que el cliente percibe e interactúa. Las vistas no tienen interacción directa con los modelos.

Los **controladores** se encargan de interconectar los modelos y las vistas. Los controladores utilizan a los modelos para manejar los datos. Luego también pueden pasar estos datos a las vistas para ser mostradas a los clientes. Son, por tanto, la lógica intermedia entre los otros dos componentes.

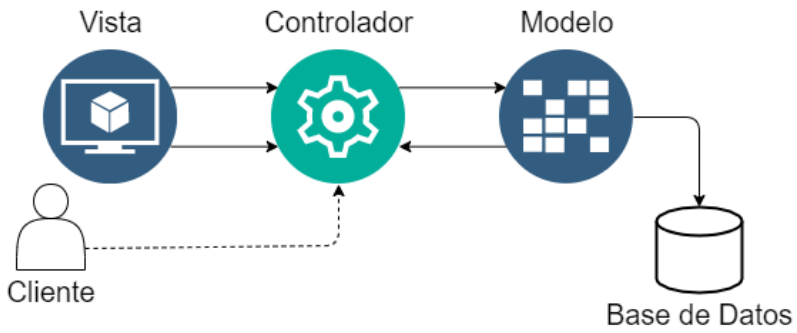

Figura 8. Arquitectura MVC

3.3.2 Asincronismo

La asincronía es un pilar fundamental a la hora de crear un servidor. Cuando una tarea requiere la finalización de otra previamente es donde entra en juego este concepto. Es necesario un mecanismo que garantice que una tarea que consume tiempo extra en procesarse ha terminado. En JavaScript se encuentran varios patrones asíncronos:

Una función callback es aquella que se pasa como argumento a otra función y es invocada una vez la operación asíncrona ha terminado. Este tipo de funciones pueden anidar en sí mismas más funciones callback, escalando sus funcionalidades, así como la funcionalidad de trabajar con ellas.

Las promesas son objetos que nos indican el estado de una tarea asíncrona. Pueden encontrarse en tres estados: cumplida, rechazada o pendiente. Una vez una promesa ha finalizado, con los bloques .then(), .catch() y .finally() podemos invocar los callbacks correspondientes. Además, estos bloques son anidables y van sucediendo de manera secuencial, por lo que si hay dos bloques .then() seguidos, el segundo no se ejecutará hasta que el primero haya terminado. Para crear una promesa hay que indicar con los métodos resolve o reject el resultado de la promesa, siendo el primero para indicar que ha sido cumplida y el segundo rechazada. Cuando una promesa ha sido rechazada podemos atrapar el rechazo con el bloque .catch() y ejecutar otra función callback. El bloque .finally() se ejecutará siempre, independientemente de que haya sido cumplida o rechazada.

Las promesas suponen una forma muy agradable y simplificada de trabajar con asincronismo.

Async/await es otra propuesta de JavaScript para trabajar con asincronismo. La palabra reservada async permite declarar una función como asíncrona, haciendo que esta siempre devuelva una promesa. La palabra reservada await espera hasta que la promesa es resuelta. Await solo puede ser utilizada dentro de funciones async.

Trabajar con async/await permite hacer más sencillo y legible el código (al eliminar anidamientos interminables) que utilizando promesas con los bloques .then(), .catch() y .finally().

En este proyecto se ha optado por usar el modelo de promesas ya que es conceptualmente más sencillo y al no haber asincronismo de alta complejidad, no hay gran anidamiento en estos bloques y no induce ilegibilidad.

3.3.3 Almacenamiento de datos

Como prácticamente cualquier servicio, es necesario un sistema que almacene datos ya sean temporales o permanentes. Para ello se usan las bases de datos.

Entre los distintos tipos de bases de datos existentes, los más comunes son los relacionales (SQL) frente a las no relacionales (NoSQL).

Para este proyecto se ha usado la base de datos MongoDB. Esta es una base de datos de tipo NoSQL. Se ha elegido esta base de datos por las siguientes razones:

- Los datos que guarda la base de datos son pequeños.
- Apenas existen relaciones entre los datos.
- Las queries que se utilizan son sencillas, ya que, al ser ligeras, devuelven los esquemas en su totalidad.
- La integración con Node.js es excelente, contando con librerías como Mongoose que permiten que la aplicación sea desarrollada pensando en los datos y no en los modelos.
- Es fácil de usar e intuitiva.
- Buena documentación.
- Moderno.
- Ideal para entornos cuyos recursos no sean muy elevados.
- Alta escalabilidad.
- Uso optimizado al usar JSON.

MongoDB permite ser desplegado tanto en servidor local como en un clúster en la nube. En este proyecto se ha optado por el servidor local para evitar cualquier tipo de coste extra que pudiera suponer mantener un servicio *cloud*. Además, así el datacenter estará en el mismo lugar que el servidor backend y no habrá posibles problemas de latencia.

Se ha utilizado la librería Mongoose para implementar MongoDB. Mediante esta librería es posible definir ciertos 'esquemas' (Schemas) que ayudan a la hora de usar colecciones sin una estructura predefinida. Permite así una abstracción de los modelos para parecer que se está trabajando con objetos en vez de con datos.

La base de datos cuenta con tres colecciones:

Audios: En esta colección se almacenan los datos de cada petición. El nombre original del archivo, la ruta en la que se almacena y todos los metadatos de configuración enviados en el formulario como los stems, el bitrate y el códec que se quiere utilizar para la conversión.

Users: En esta colección se almacenan los datos de registro de los usuarios. El email, el hash de la contraseña y el salt que se ha utilizado para generar el hash.

Sesiones: En esta colección se almacenan las cookies de sesión que se generan cuando los usuarios inician sesión. Guarda la cookie de sesión y la fecha de expiración. Cuando la sesión expira, el documento se elimina automáticamente.

3.3.4 Motor de vista

Los Motores de Vista o View Engines son otra potente herramienta que nos permite generar código de manera dinámica. En concreto, generar HTML de forma dinámica.

Existen varios motores de plantillas distintos, como Pug o Handlebars. Cada uno cuenta con sus ventajas y desventajas. En este proyecto se ha utilizado el motor EJS.

Para configurarlo en Express basta con establecer el nombre del motor que se va a usar y el directorio en el que se van a encontrar esos ficheros:

app.set('view engine', 'ejs')

app.set('views', 'views')

La ventaja de usar EJS es que su sintaxis es igual que HTML y JavaScript. Para introducir código JavaScript basta con introducirlo en los caracteres <% >

```
router.get('/login', (req, res, next) => {
  res.render('login', {
    autenticado: false,
    path: '/login'
  });
});
```

Figura 9. Renderizado de EJS

En la Figura 9 se puede apreciar como se sirve una página dinámica. Los parámetros que se pasan son 'autenticado' y 'path' y la plantilla se encarga de insertarlo en el código y renderizarlo. Finalmente, se le envía al cliente que ha solicitado el recurso.

Capítulo 4. Desarrollo de la aplicación

4.1 Discusión de alternativas disponibles

Tensorflow.js es una librería para el aprendizaje automático en JavaScript. Esta librería permite desplegar redes tanto en el backend como en el frontend.

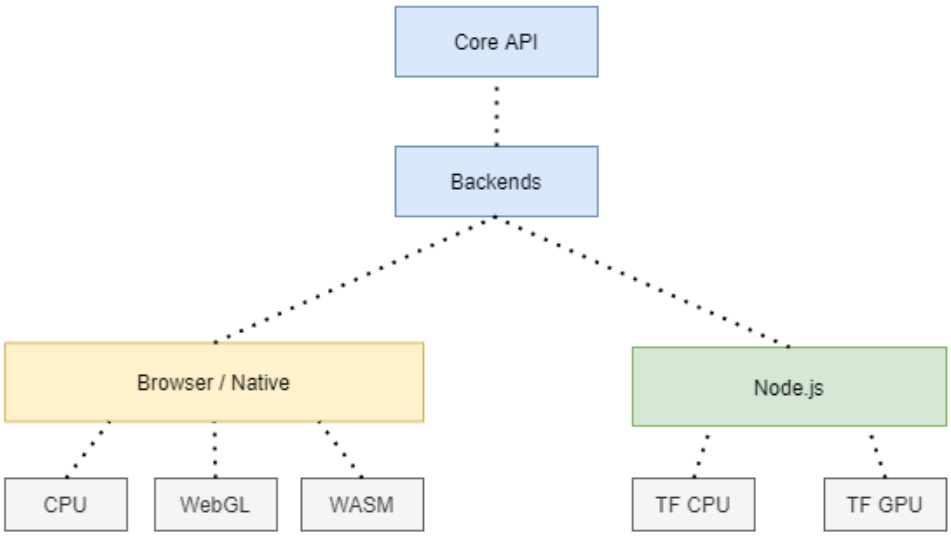

Figura 10. TensorFlow.js (Ann Yuan, 2020)

La Figura 10 muestra los distintos backends que se pueden utilizar con Tensorflow.

Una red en el frontend delegaría la carga computacional de la red al navegador del cliente, reduciendo así la carga de trabajo del servidor backend.

Se pueden encontrar limitaciones, como por ejemplo que el modelo se ha de enviar al cliente antes de que lo ejecute, y hay modelos que pueden ocupar mucha memoria. Además, muchas de estas redes son computacionalmente caras, llegando a ser muy común que se utilice la gráfica para el procesado de redes. Esto se suma a que, además, los navegadores necesitan más potencia computacional que un simple programa por lo general. Esto puede suponer una degradación del servicio.

De esta manera, se podría unificar todo el backend y el servicio de Deep Learning con JavaScript.

Esta librería nos aporta varias opciones como son:

- Crear nuestra red neuronal directamente en Tensorflow.js

- Convertir un modelo de Python nativo a Tensorflow.js

4.1.1 Red nativa Tensorflow.js

Una de las posibilidades a la hora de implementar una red neuronal mediante un servicio web sería crear directamente la red nativamente mediante Tensorflow.js.

En este caso, la red neuronal ya esta construida y entrenada en Python, por lo que construirla de nuevo para Tensorflow.js no es una opción viable.

4.1.2 Convertir modelo pre-entrenado a Tensorflow.js

La siguiente opción consiste en convertir el modelo pre-entrenado de Tensorflow a Tensorflow.js. Sin embargo, Tensorflow.js aún no cuenta con todas las operaciones que permite Tensorflow (como el tf.Estimator) y que esta red utiliza, por lo que aún no puede ser plenamente convertida.

En un futuro en el que se agreguen todas las operaciones disponibles de Tensorflow a Tensorflow.js sí podrá convertirse cualquier modelo y podría valorarse esta alternativa.

4.1.3 Servir modelo nativo pre-entrenado

Otra opción por la que se puede optar es utilizar Node.js como backend. De esta manera, se nos ofrecen dos librerías por las cuales se puede ejecutar nativamente una red de Tensorflow en un backend JavaScript que utilice Node.js como tiempo de ejecución como es este caso. La diferencia entre TF CPU y TF GPU radica en que TF GPU acelera las operaciones con la gráfica del sistema y TF CPU únicamente con la CPU.

Conceptualmente, este modelo de integración supone comodidad en el desarrollo y un funcionamiento adecuado. Primero se desarrolla cómodamente la red en Python, ya que es el lenguaje inspirado para ello y luego tan solo hace falta exportar el modelo en el formato adecuado que permite a la librería de Node.js leerlo.

4.1.4 Proceso en segundo plano

Como última opción se presenta la posibilidad de crear un subproceso que llame a Python y este ejecutase la red de Tensorflow, devolviendo al finalizar el subproceso los resultados obtenidos de vuelta a Node.

La desventaja de este método es la posible dificultad que puede generar estar trabajando con subprocesos y, a la vez, el rendimiento del sistema puede verse comprometido al estar trabajando con distintos entornos y teniendo que intercambiar la información entre procesos distintos.

Además, cada vez que se llama al subproceso, Python debe cargar el modelo nuevamente por completo. Este es uno de los puntos que más tiempo tardan de todo el proceso.

4.2 Implementación del servicio web

La aplicación web tiene dos funcionalidades. Por una parte, el uso de la red, que se puede subdividir en tres procesos desde el punto de vista del servidor:

- Recibir el archivo y almacenarlo
- Pasar el archivo por la red para obtener las pistas separadas
- Comprimir las pistas y enviárselas al cliente

Por otra parte, está el sistema de cuentas de usuario que por ahora no tiene relación alguna con la red y será explicado más tarde.

Para el desarrollo se han desarrollado los tres procesos mencionados anteriormente por separado como módulos y luego se han juntado todos en el controlador de la petición.

4.2.1 Servidor

El archivo *app.js* es el archivo raíz de la aplicación. Este es el que va a estar continuamente escuchando al puerto y recibiendo todas las llamadas. Cada petición irá recorriendo el flujo de este archivo, yendo por unas u otras rutas según se ha establecido, atravesando los correspondientes middlewares. En este archivo también es donde se establece la conexión con la base de datos.

Para manejar las peticiones se ha usado la clase Router de Express. De esta manera, conseguimos delegar el control de las rutas al archivo que indiquemos. Este archivo solo será encargado de enviar cada petición al controlador correspondiente.

```
// /main → GET
router.get('/main', (req, res, next) ⇒ {
  res.render('main', {
    autenticado: req.session.logged,
    path: '/main'
  });
});

// /signup → GET
router.get('/signup', (req, res, next) ⇒ {
  res.render('signup', {
    autenticado: false,
    path: '/signup'
  });
});

// /login → GET
router.get('/login', (req, res, next) ⇒ {
  res.render('login', {
    autenticado: false,
    path: '/login'
  });
});

// /main → POST
router.post('/main', controlador.fullFunction);

// /signup → POST
router.post('/signup', check('email').isEmail(),userController.signup);

// /login → POST
router.post('/login', check('email').isEmail(), userController.login);

// /logout → POST
router.post('/logout', auth.ensureAuth, userController.logout);
```

Figura 11. Rutas

En la Figura 11 se puede ver como se establecen el método (GET o POST) y el path (/login) y luego se añade una función que controla la petición, el conocido Controlador del patrón MVC. En el caso de los GET solo se trata de renderizar la página dinámica correspondiente. El resto están en módulos separados ya que hacen uso de funciones más complejas.

Como se ha mencionado anteriormente, el primer paso es recibir la solicitud del cliente y almacenar la pista de audio. Para ello se ha utilizado Multer, un paquete que, a modo de middleware, permite almacenar el archivo entrante en la ubicación que le indiquemos (en este caso, una carpeta llamada audios) y con el nombre que le asignemos (el nombre original junto a un valor numérico aleatorio generado con la fecha en ese instante).

El siguiente paso es procesar el audio por la red. Este bloque se explicará en el siguiente apartado pues es el más complejo.

Por último, se ha de comprimir la respuesta y enviársela al cliente. La red al finalizar devuelve una carpeta con todas las pistas separadas dentro. Esta es la carpeta que vamos a comprimir para enviar al cliente. Para esta compresión, se ha utilizado otra librería llamada 'archiver' que nos permite comprimir e ir streameando la respuesta poco a poco conforme vaya estando lista.

4.2.2 Implementación de la red

El primer paso de todos es obtener el modelo pre-entrenado. Al descargar el repositorio y hacer una primera inferencia, los modelos ya entrenados son descargados y usados por la red.

No obstante, el formato en el que vienen estos modelos no es el adecuado, ya que se quiere en formato .pb para poder cargarlo en Node.js.

Para solucionar esto, se ha utilizado un script que exporta el modelo al formato que se desea.

Una vez está listo, el modelo es copiado en el repositorio para ser usado posteriormente.

A continuación se carga el modelo y mediante el método getMetaGraphsFromSavedModel se obtienen los metadatos de la red (los inputs y outputs).

```
modelInfo [ { tags: [ 'serve' ], signatureDefs: { serving_default: [Object] } } ]
tags [ 'serve' ]
signatureDefs {
  serving_default: {
    inputs: {
      audio_id: [Object],
      mix_spectrogram: [Object],
      mix_stft: [Object],
      waveform: [Object]
    },
    outputs: { accompaniment: [Object], audio_id: [Object], vocals: [Object] }
  }
}
```

Figura 12. Meta Graphs del modelo

Como se puede ver en la Figura 12, a los metadatos no basta con pasarle el fichero de audio a la red, sino que hay que realizar un preprocesamiento del archivo. En concreto para la función de separar solo haría falta la waveform, ya que los otros inputs (espectrograma y stft) solo son usados por las otras funciones de entrenamiento y evaluación de la red.

La waveform ha de estar en un formato de datos concreto. Tras analizar la red se ha visto que el formato adecuado es 32-bit flaoting-point little-endian (f32le), que es un array bidimensional en punto flotante de 32 bits. Para ello se va a utilizar ffmpeg, guardando el resultado de la conversión en un buffer para ir encanalando los datos hacia la red de chunk en chunk.

```
var command = ffmpeg(readStream)
  .addOption('-ar', '44100')
  .addOption('-ac 2')
  .format('f32le')
  .on('start', (commandLine) => {
    console.log('FFmpeg Command:' + commandLine);
  })
  .on('error', (err, stdout, stderr) => {
    console.log('An error occurred: ' + err.message);
  })
  .on('end', (stdout, stderr) => {
    var returnedText = stderr;
    console.log(returnedText);
```

Figura 13. Configuración comando FFMPEG

Una vez el modelo está cargado correctamente y los datos están siendo encaminados al modelo surge un problema.

```
Error: Session fail to run with error: audio_id:0 is both fed and fetched.
    at NodeJSKernelBackend.runSavedModel (C:\Users\Luis\Desktop\TFG\Prisma\node_modules\@tensorflow\tfjs-node\dist\nodejs_kernel_backend.js:446:43)
    at TFSavedModel.predict (C:\Users\Luis\Desktop\TFG\Prisma\node_modules\@tensorflow\tfjs-node\dist\saved_model.js:362:52)
    at C:\Users\Luis\Desktop\TFG\Prisma\controllers\modelController.js:30:30
```

Figura 14. Error modelo

La red tiene como input y output referenciada la misma variable audio_id. Se ha intentado resolver este error antes de exportar los modelos de Python al formato compatible con Node, no obstante, esto no es posible sin cambiar la red y esto podría suponer cambiar el funcionamiento global o incluso tener que reentrenar los modelos, por lo que no ha sido posible solucionarlo.

Para sobrepasar esta dificultad, se ha optado por implementar la red como un subproceso de Python. Para ello se ha utilizado la librería nativa de Node.js 'child-process' que nos permite generar subprocesos.

```
const python = spawn('python', ['python_scripts/spleeter/__main__.py', 'separate', '-i', audioPath, '-p', stems, '-c', codec, '-b', bitrate, '-o', 'audio_separated']);
python.stdout.on('data', function (data) {
    console.log('Pipe data from python script ...');
    dataToSend = data.toString();
});
python.on('close', (code) => {
    console.log(`child process close all stdio with code ${code}`);
    console.log('Finalizado')
    resolve([code, path.parse(audioFile.title).name,path.parse(audioPath).name]);
});
```

Figura 15. Subproceso de Python

En la Figura 15 se puede apreciar como se genera el proceso igual que si se ejecutase la red en la cli de un ordenador, pasándole los correspondientes parámetros de configuración. Cabe destacar que todo este proceso es asíncrono por lo que involucra a una promesa que se resuelve cuando el subproceso de Python finaliza.

4.2.3 Signup y login

Se ha creado también un sistema de registro de usuarios que por ahora no afecta a la hora de utilizar la página web de manera convencional. En este aspecto está pensado para un desarrollo futuro.

Cuando se inicia sesión satisfactoriamente se genera una sesión con un parámetro logged=true para poder guardar el estado del usuario. La librería 'express-session' ayuda a establecer la cookie de sesión de forma sencilla.

Cuando se establece una gran cantidad de sesiones de forma simultánea, el servidor no puede guardarlas todas en memoria. Es buena praxis almacenar estos datos de sesión dentro de la base de datos. Además, estos documentos son eliminados una vez la cookie de sesión es destruida por el cliente o expira por la fecha establecida.

Para lo que sí es necesario registrarse es para utilizar la API REST del servicio, que solicitará unas credenciales (token web) que se consiguen mediante este registro.

4.2.4 Interfaz de usuario

Para la interacción del usuario se ha creado una web intuitiva y llamativa. Se ha realizado únicamente con HTML dinámico (EJS), CSS para el aspecto visual y JavaScript para la interacción con el DOM.

Figura 16. Página de inicio

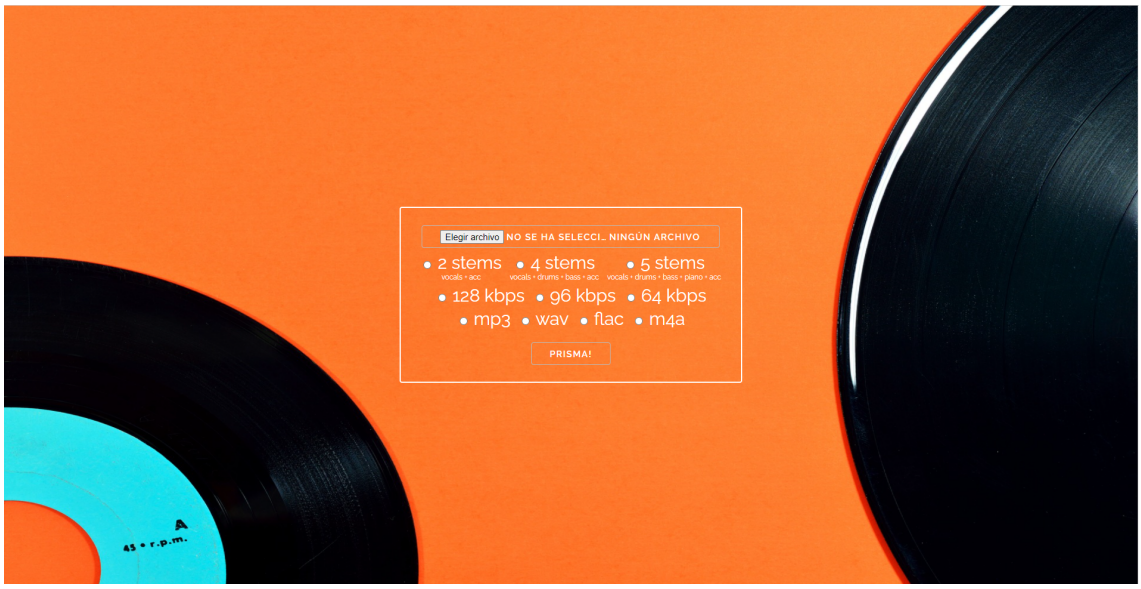

Figura 17. Página de inicio 2

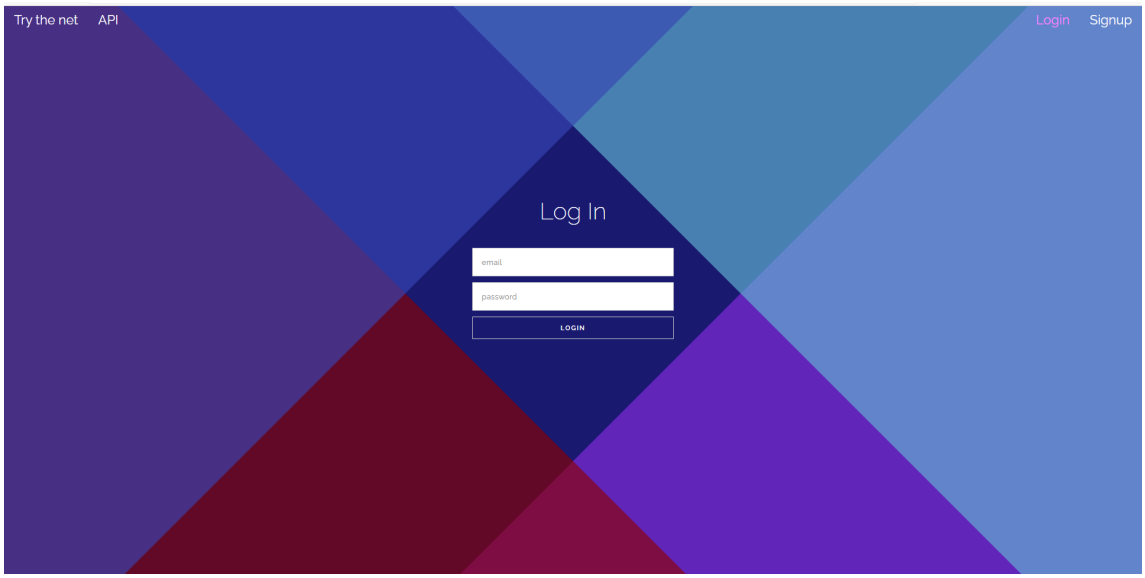

Figura 18. Página de login

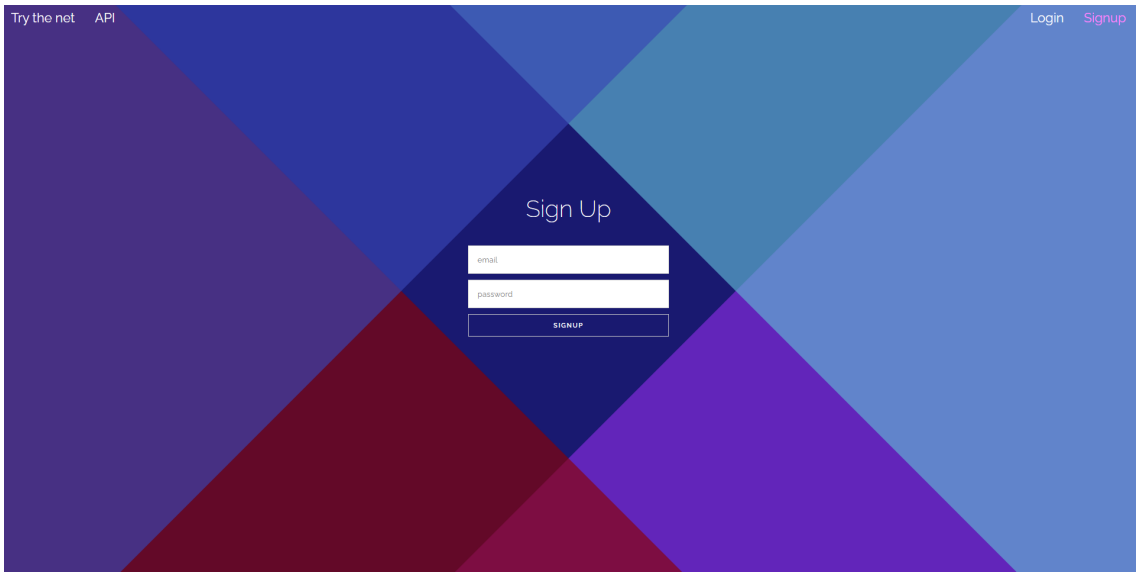

Figura 19. Página de signup

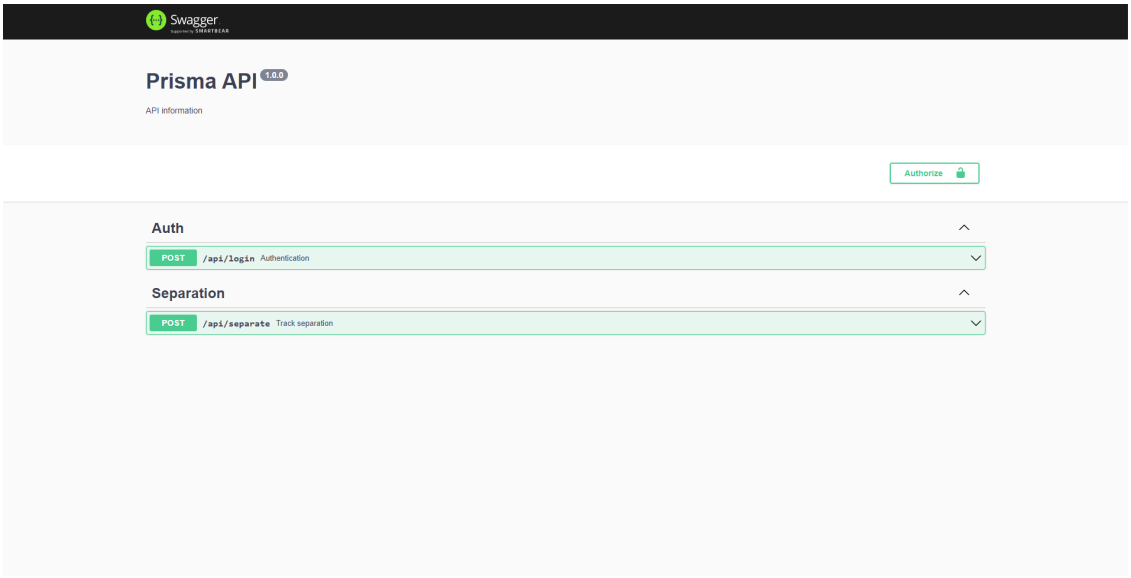

Figura 20. Página API

4.2.5 Descripción funcional

Para utilizar la aplicación hay que entrar en la página de inicio (Try the net) y bajar hasta el segundo panel. En este aparece un formulario en el que el cliente elegirá el archivo que desea convertir junto a la configuración deseada.

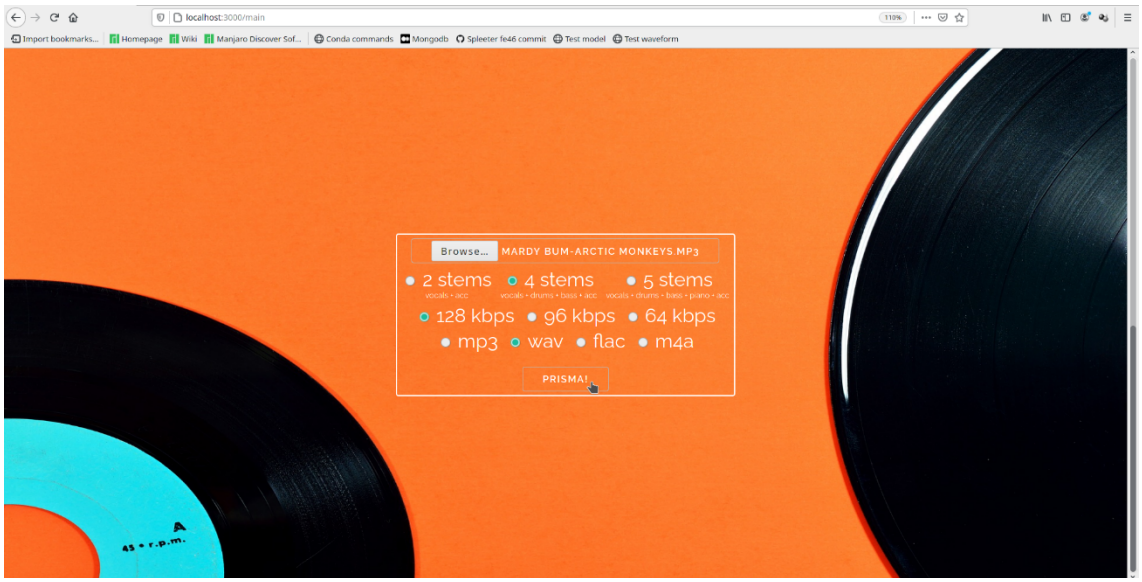

Figura 21. Elegir configuración

Esto envía servidor la petición y comenzará a procesarla. Una vez haya finalizado, enviará de vuelta al cliente un archivo zip.

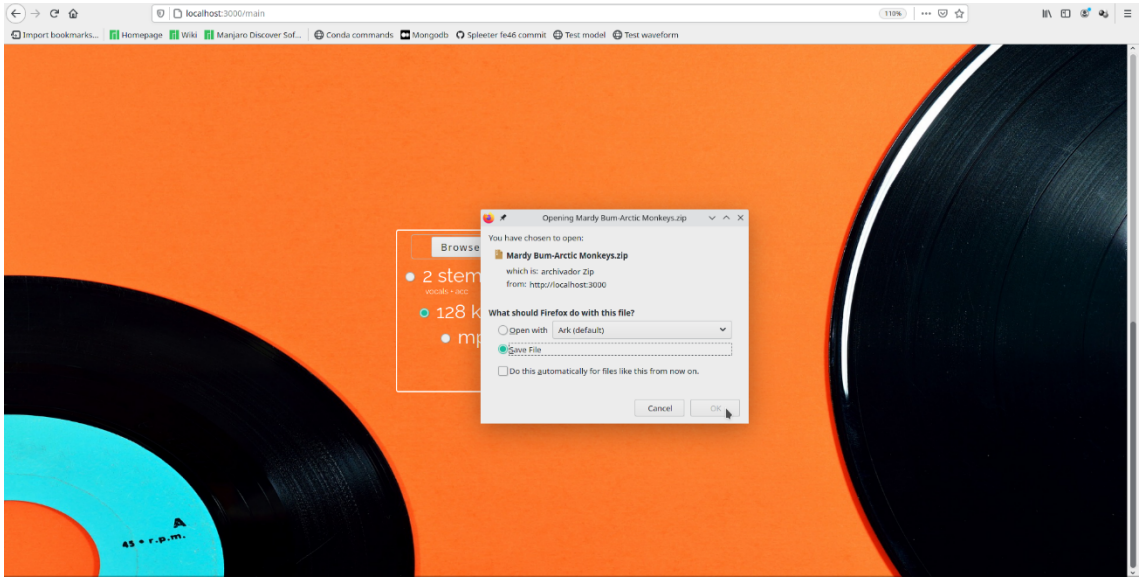

Figura 22. Archivo comprimido

Una vez se descomprime el archivo, se pueden ver las pistas con la configuración que se habían elegido.

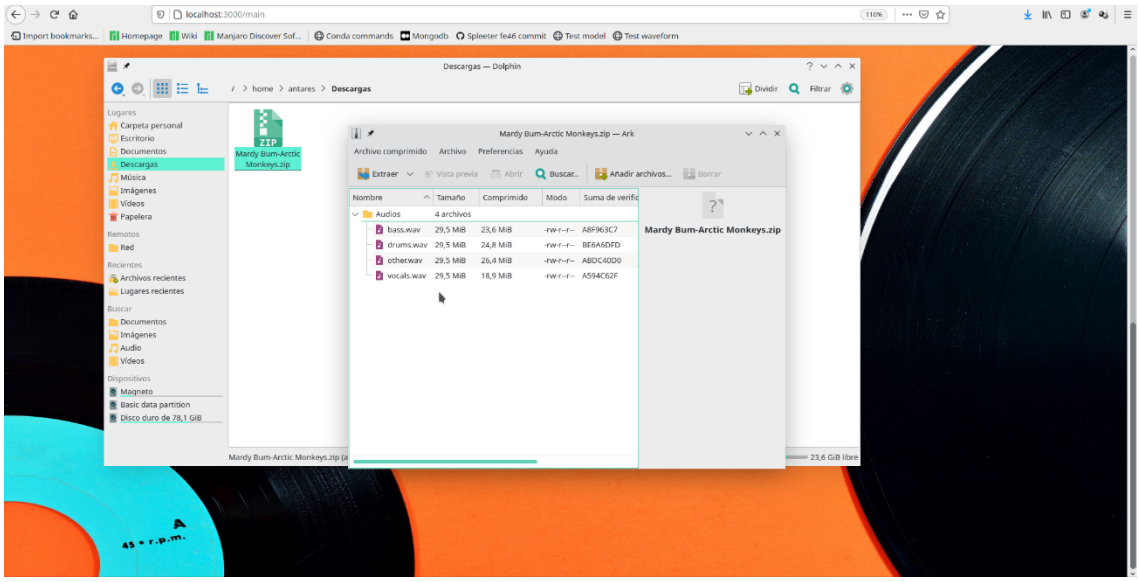

Figura 23. Contenido del archivo comprimido

En la Figura 23 se puede ver que están los cuatros archivos en el formato que habíamos indicado en el formulario.

4.3 REST

Este servicio ya sería funcional accediendo a la página web habitual y haciendo uso de ella. Sin embargo, hay otros usos que no requieren de esta interfaz de representación (HTML), sino que solo existe interés en hacer uso del servicio. Esto es, obtener los datos. Por ello existe otra arquitectura para lograr esto. Esta arquitectura es conocida como REST (Representational State Transfer) donde se deja de lado la visualización de los datos y estos pasan a estar en primer plano.

Esta arquitectura se sostiene sobre los siguientes principios (Al-Zoubi & Wainer, 2009):

- Cliente-Servidor. El cliente y el servidor deberán ser independientes entre sí, siempre y cuando la interfaz entre ellos no cambie.
- Sin estado (Stateless). Todas las peticiones son independientes, por lo que no habrá sesiones ni historial. Cada petición será tratada como si fuera la primera. Si se quiere establecer algún tipo de estado, esa información deberá ser incluida en todas las peticiones. Por ejemplo, un token de autenticación.
- Interfaz Uniforme. Todos los componentes siguen rigurosamente las mismas reglas.
- Cacheable. Se debe cachear los recursos siempre y cuando sea posible y deberán ser declarados como cacheables. Esto se puede implementar tanto en el cliente como en el servidor.
- Sistema por capas. El cliente puede no saber si está conectado directamente a un endserver o a uno intermedio.
- Código bajo demanda (opcional). REST permite solicitar código para ser ejecutado. Sin embargo, esto debe ser opcional ya que el cliente podría no ser capaz de descargar o ejecutar este código y no se debería depender de esto.

Siguiendo estos principios se han planificado dos endpoints para hacer uso del sistema como se pueden apreciar en la Figura 24.

Figura 24. Endpoints

El primer endpoint lo tenemos en el URI: /api/login y sirve para autenticarse y conseguir de esta manera el token que va a permitir hacer uso del otro endpoint.

El segundo endpoint lo encontramos en el URI: /api/separate. Aquí se envía el archivo de audio junto con las configuraciones de la separación. Devuelve las pistas ya separadas.

4.3.1 Endpoints

4.3.1.1 Login

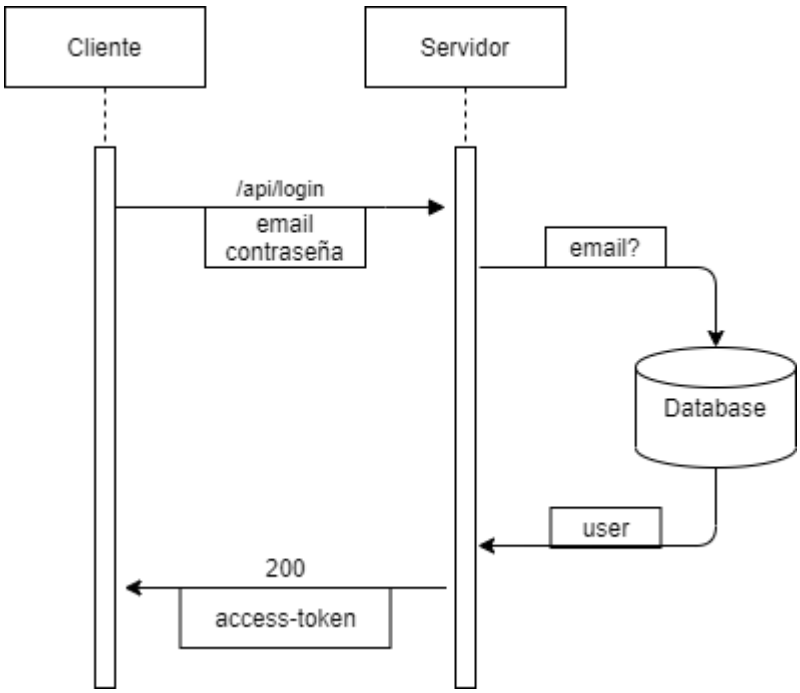

Figura 25. Login Endpoint

Este endpoint define un recurso para la autenticación de las credenciales y devuelve un token. Con este token se podrá hacer uso del resto de endpoints.

```
if (user.validPassword(pw, user)){
    const jwtoken = jwt.sign({
        id: user._id.toString()
    }, 'fideo kojima', {
        expiresIn:'1h'
    });
    return res.status(200).json({token:jwtoken, id: user._id.toString()})
}
else {
    return res.status(401).send({
        message : "Wrong Password"
    });
}
```

Figura 26. REST login

En la Figura 26 se puede ver el código principal del funcionamiento. Se calcula el hash de la contraseña enviada y se compara con la almacenada en la base de datos de dicho usuario. Si coincide, se genera un json web token mediante jwt.sign() y una clave secreta. Además, se establece la duración de dicho token. En este caso, de una hora. Este token se le envía de vuelta al cliente junto con el id de usuario.

4.3.1.2 Separate

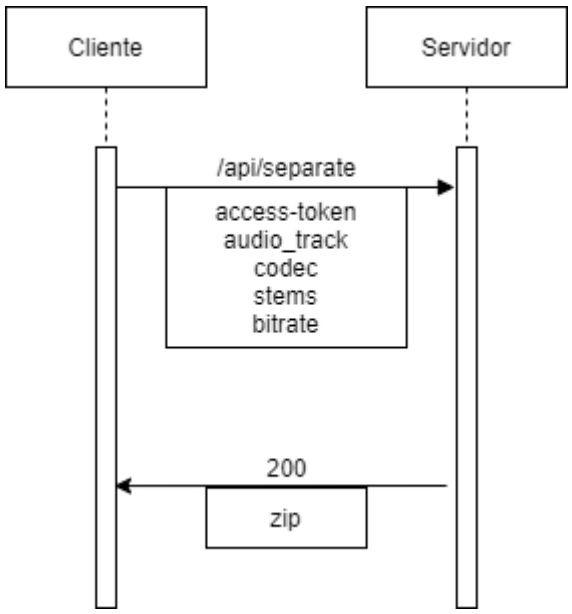

Figura 27. Separate Endpoint

Este endpoint define el recurso que devuelve las pistas de audio separadas. Recibe los datos de configuración opcionales (codec, bitrate y stems) junto con el archivo de audio. Para que este recurso funcione hay que adjuntar el web token en la cabecera de la petición.

Cuando la petición llega, la maneja primero un middleware que se encarga únicamente de validar el token de acceso.

```
exports.validateToken = (req, res, next) => {
    const wtoken = req.headers['access-token'];
    console.log(wtoken)
    let decToken;
    try{
        decToken = jwt.verify(wtoken, 'fideo kojima')
    } catch(err){
        console.log(err);
    }
    if(!decToken){
        console.log('No autenticado');
        return res.status(401).send({
            message: 'Authentification missing'
        })
    }
    else{
        console.log('Token validado');
        next();
    }
}
```

Figura 28. Middleware validación de token

Si el token no está correcto, se envía un error 401 que indica la carencia de credenciales de autenticación. Si el token es correcto, pasa a ser manejado por el controlador del recurso. A partir de aquí funciona exactamente igual que el servicio web.

4.3.2 CORS

Para que el servicio REST funcione se ha de controlar el acceso HTTP. El Intercambio de Recursos de Origen Cruzado (CORS) es un mecanismo de cabeceras que permite que un usuario tenga permisos para acceder a recursos desde un servidor distinto al dominio al que este pertenece.

Por defecto, las peticiones son restrictivas y no permiten esto. Para permitir esto, se van a utilizar cabeceras CORS solo para los endpoints del REST.

Para ello, se ha creado una función middleware que al ser recorrida añade estas cabeceras a la respuesta.

```
exports.corsHeaders = (req, res, next) => {
    res.setHeader('Access-Control-Allow-Origin', '*');
    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, PATCH, DELETE, OPTIONS');
    res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
    next();
}
```

Figura 29. Middleware cabeceras CORS

En la Figura 29 se puede ver como las cabeceras que se establecen son:

- Acces-Controll-Allow-Origin. Con el valor wildcard se permite que se pueda acceder desde cualquier origen a los recursos.
- Access-Control-Allow-Methods. En esta cabecera se declaran los métodos que pueden ser enviados al servidor. Pese a que este servicio solo utiliza POST, se ha puesto el resto para cuando se añadan más endpoints.
- Access-Control-Allow-Headers. Esta cabecera indica que encabezados HTTP se pueden utilizar en la solicitud.

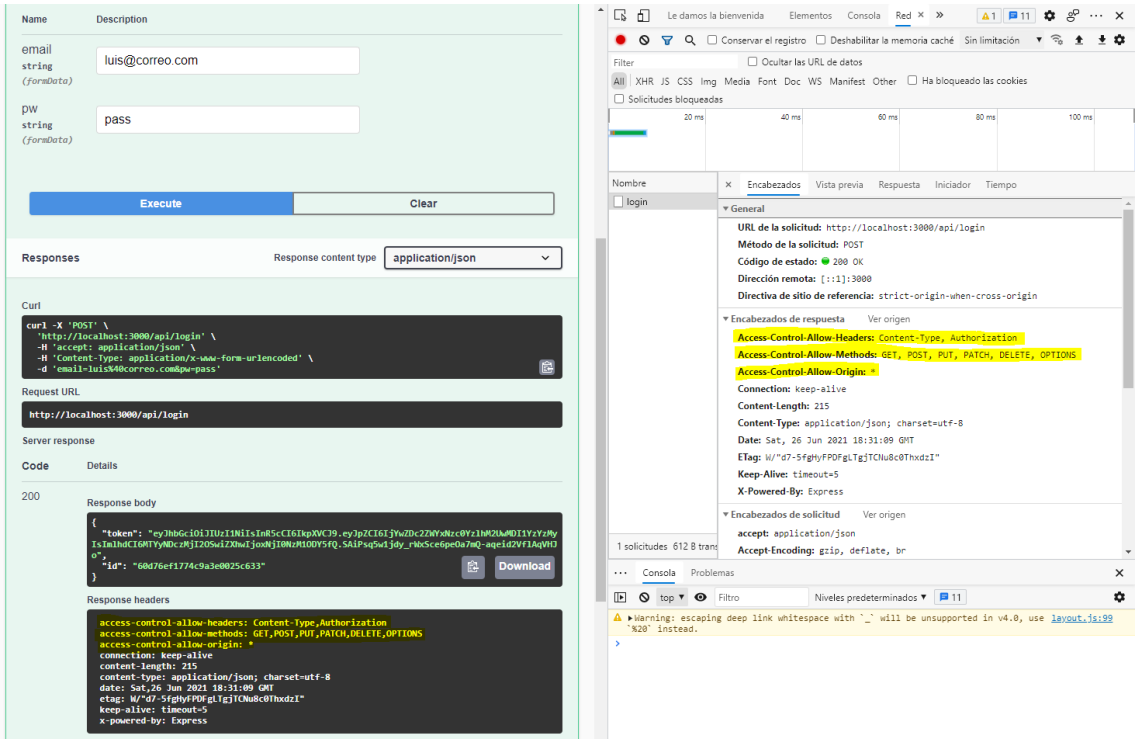

Figura 30. Cabeceras CORS en respuesta

En la Figura 30 se puede ver que al lanzar la petición a la ruta /api/login aparecen las cabeceras que se han establecido antes y que permiten acceder a este recurso.

4.3.3 Documentación

Para que un cliente no tena problemas a la hora de comprender y emplear adecuadamente una API, esta debe ser correctamente documentada. Para ello existen tecnologías especializadas en ello. En este caso se ha utilizado Swagger.

Swagger es una herramienta que permite construir una documentación rica y funcional, permitiendo interactuar con los diferentes endpoints en forma de prueba. Esto ayuda al cliente a comprender el funcionamiento y a visualizar la solicitud por completo.

Para utilizar Swagger se han utilizado dos dependencias:

- swagger-jsdoc
- swagger-ui-express

El funcionamiento es sencillo, solo se ha tenido que inicializar las dependencias y escribir las anotaciones mediante el lenguaje YAML.

```
/**
 * @swagger
 * /api/login:
 *  post:
 *      summary: Authentication
 *      tags: [Auth]
 *      description: Route to log in with the credentials
 *      consumes:
 *       - application/x-www-form-urlencoded
 *      parameters:
 *       - in: formData
 *         name: email
 *         type: string
 *       - in: formData
 *         name: pw
 *         type: string
 *      responses:
 *          '200':
 *              description: Authentication succesful
 *          '400':
 *              description: User not found
 *          '401':
 *              description: Wrong password
 *
 */
```

Figura 31. Anotaciones en YAML

Como se puede ver en la Figura 31 solo se ha de rellenar toda la información en las anotaciones y las dependencias instaladas harán el resto, renderizando el código y sirviendo una página interactiva. El resultado es el siguiente:

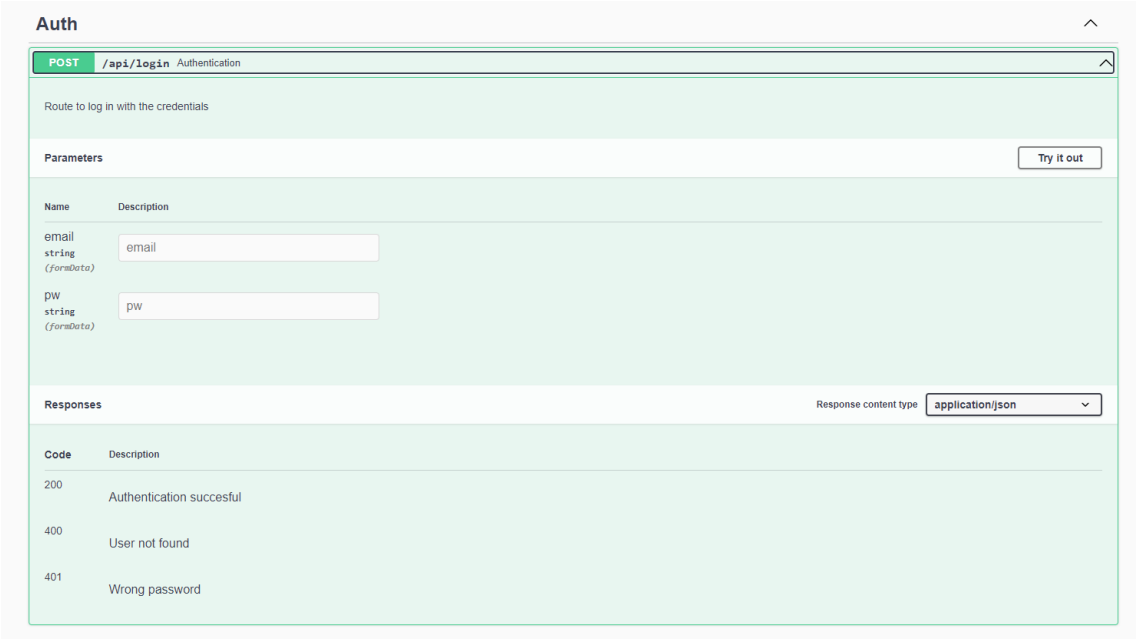

Figura 32. Documentación Login

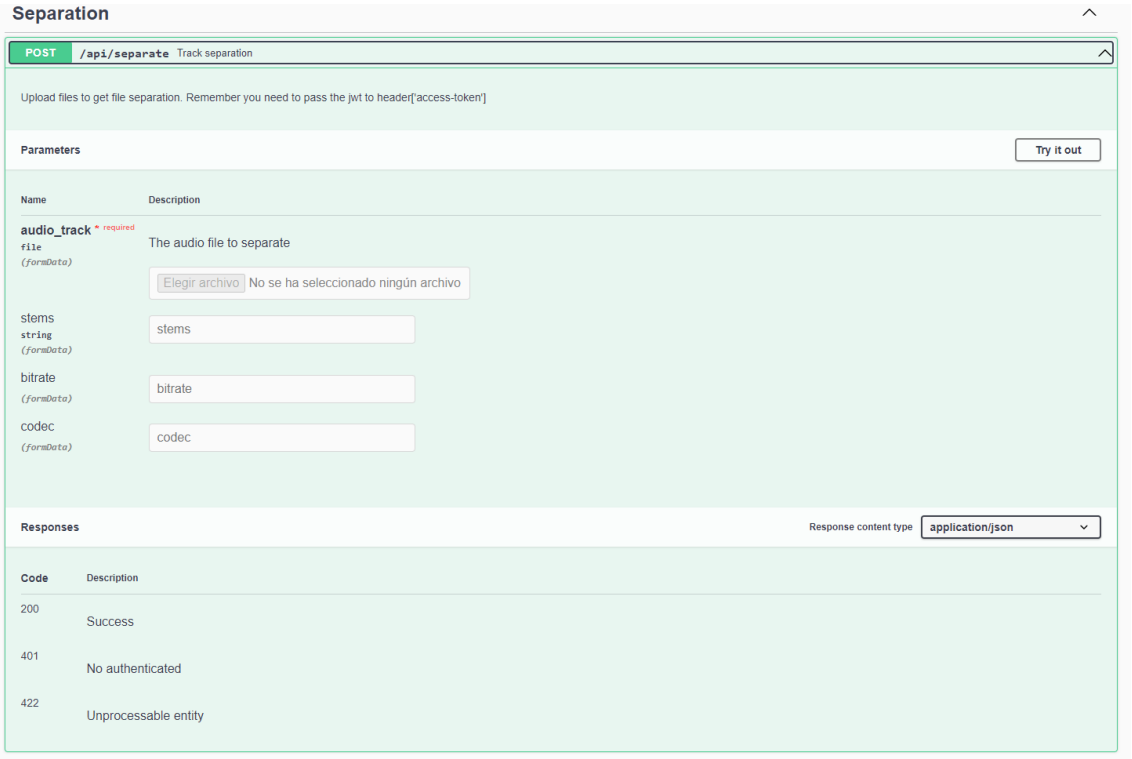

Figura 33. Documentación Separate

La mayor ventaja es que permite rellenar los campos de las peticiones y visualizar la propia respuesta del servidor. Además, aporta toda la información necesaria para su uso correcto, como las descripciones de los parámetros o los posibles códigos de estado de la respuesta junto a su descripción.

4.4 Seguridad

La seguridad es un pilar fundamental en cualquier servicio web. La finalidad es prevenir cualquier ataque o vulnerabilidad hacia nuestra aplicación.

En lo referente a la seguridad de los datos, MongoDB ya proporciona sus propios mecanismos de seguridad. Encripta los datos, encripta las comunicaciones, contiene roles de acceso y limita la exposición a la red entre otros. Por ello, se considera que los datos están bien protegidos ante posibles amenazas.

Respecto a los propios datos almacenados, la única información sensible es la colección de usuarios, en la cual están almacenados los datos personales de cada usuario. Sin embargo, para no tener información susceptible, se guarda el hash de la contraseña junto al *salt* que lo generó en vez de la propia contraseña. Este es el método adecuado de guardar credenciales ya que, si la información se ve comprometida, el atacante nunca sabrá la contraseña real, gracias a las propiedades de los hashes.

Otro punto importante en cuanto a la seguridad es validar los datos que son enviados desde el cliente. Esto se puede hacer opcional por parte del cliente, pero este puede saltarse esta validación y enviar cualquier tipo de datos sin validar al servidor. Por ello, es crucial añadir validación en el servidor. Para ello se ha utilizado la dependencia de express-validator, que añade un middleware que permite validar todos los datos de entrada.

```
router.post('/signup', check('email').isEmail(),userController.signup);
```

Figura 34. Middleware Validación

En la Figura 34 se puede ver como añadimos el middleware de validación antes de entrar al controlador. Así, cuando la petición sea manejada por este, ya tendrá validado todos los campos del registro. En este caso, verifica que el campo email de registro tiene el formato correcto de correo electrónico.

Las cabeceras, cuando se usan bien pueden proporcionar robustez en cuanto a la seguridad. En este caso se ha modificado la cabecera de 'X-powered-by' ya que por defecto tenía el valor de 'Express'. Por ello, para que no ofrezca información al cliente sobre qué backend se esta usando y pueda buscar vulnerabilidades específicas se ha modificado el valor.

También se ha creado otro middleware para establecer la cabecera de Content Security Policy. Con esta cabecera se especifican las fuentes de las que el contenido puede ser cargado. Por lo tanto, cualquier código que no provenga de las fuentes marcadas en esta cabecera será bloqueado para que no se ejecute. Esta cabecera dificulta a los atacantes inyectar código con ataques de tipo XSS.

```
Content-Security-Policy: default-src 'self'; font-src 'self' https://fonts.gstatic.com; img-src 'self'; script-src 'self';
style-src 'self'; frame-src 'self'; style-src-elem 'self' https://fonts.googleapis.com
```

Figura 35. Cabecera CSP

Se puede ver en la Figura 35 como solo se permite el código proveniente del propio servidor o de googlefonts para las fuentes CSS.

En cuanto al REST, el método de autenticación es con un json web token. Este token está cifrado por una clave secreta que solo el servidor conoce. Además, está configurado para que el token expire tras una hora. Esto sirve por si el token de un usuario fuera robado, perdiera validez en un tiempo relativamente corto y no pudiera seguir suplantando la identidad de el usuario.

La mayor vulnerabilidad que se puede encontrar en esta aplicación web tiene que ver con la subida de archivos al servidor. Cualquier atacante podría intentar subir código malintencionado para que sea ejecutado más tarde en el servidor. Por ello, se ha programado un filtro que solo permite el almacenamiento de archivos cuyo mimetype sea mpeg o wav (audio).

```
const filtroMime = (req, file, callback) => {
    console.log(file.mimetype);
    if (file.mimetype === 'audio/mpeg' || file.mimetype === 'audio/wav'){
        console.log("Multer: Archivo almacenado");
        callback(null, true);
    } else{
        console.log("Multer: Error al almacenar");
        callback(null, false);
    }
}
```

Figura 36. Filtro Mimetype

Capítulo 5. Despliegue

5.1 Alojamiento

Para alojar el servicio web, se ha optado utilizar un servidor virtual privado que provee la empresa Digital Ocean. Existen numerosas empresas que ofrecen este tipo de servicios como Amazon AWS o Microsoft Azure. Se ha elegido Digital Ocean por ofrecer unos servicios económicos, con multitud de funciones y configuraciones y una interfaz intuitiva y sencilla de utilizar.

Al crear el servidor virtual llamado droplet se ha elegido la distribución Ubuntu por ser una de las más sencillas que había. Para conseguir el funcionamiento del servicio hay que instalar todo lo necesario. Para ello se ha instalado mediante la herramienta 'apt' install:

- Mongodb
- Nodejs
- Npm
- Git
- Ffmpeg

La versión de Python que se necesitaba no estaba en los repositorios, por lo que se tuvo que descargar el código fuente comprimido y compilarlo mediante la utilidad 'make'.

El proyecto se clonó de GitHub y se ejecutó el comando 'npm install' para instalar todas las dependencias de Node y también se instalaron mediante el gestor de paquetes de python 'pip' todas las librerías que necesita la red para funcionar.

Una vez todo está instalado se ejecutó *app.js* para dejar el proceso activo permanentemente. Ahora, entrando en la dirección IP del droplet y al puerto indicado a través del navegador, se puede acceder sin problemas a la aplicación web y hacer uso de ella.

5.2 Dominio

Acceder a una página web mediante una dirección ip y un puerto concretos es engorroso y puede reducir el flujo de visitas. Por ello se ha obtenido un dominio web de un registrador. Estas entidades son las que gestionan los dominios.

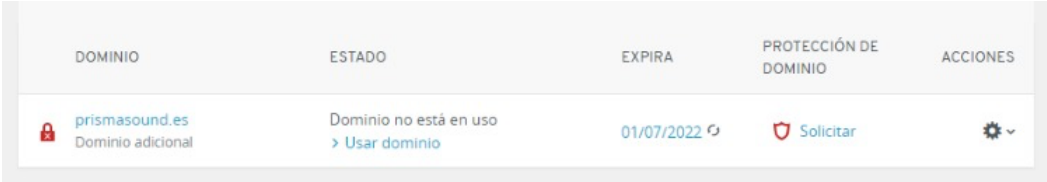

Figura 37. Dominio web

Una vez se ha adquirido el dominio, la mayoría de los registradores permiten que se gestione a través de la web de hosting. En este caso se ha hecho esto, por lo que se han agregado los servidores de nombre del servicio de alojamiento en los servidores DNS del dominio como se puede ver en la Figura 38.

Figura 38. Servidor de nombres

DNS records

Type	Hostname	Value	TTL (seconds)	
A	prismasound.es	directs to 138.68.84.161	3600	More
NS	prismasound.es	directs to ns3.digitalocean.com.	1800	More
NS	prismasound.es	directs to ns2.digitalocean.com.	1800	More
NS	prismasound.es	directs to ns1.digitalocean.com.	1800	More

Figura 39. Registros DNS en hosting

Finalmente, en los ajustes del servidor privado se ha enlazado el dominio *prismasound*.es con la dirección IP de la máquina, como se puede ver en la Figura 39. Nótese que aquí aparecen los servidores de nombre que hemos puesto en la configuración del dominio en el registrador.

5.3 Coste

El coste de mantener un servicio de este tipo vendría del alquiler mensual del servidor virtual junto con el precio del dominio. Estos precios pueden fluctuar mucho dependiendo del tipo de servidor que se elija.

Type	CPU Type	vCPUs	Memory	SSD	Transfer	Price
Basic	Shared CPU	1 vCPU	1 GB	25 GB	1 TB	$5/mo $0.007/hr

Figura 40. Coste servidor virtual

En este caso, se ha elegido la configuración más básica de todas, que cuenta con una CPU compartida, 1 GB de memoria RAM, 25 GB de almacenamiento y cuesta 5€/mes.

Los precios de los dominios también pueden variar enormemente de precio dependiendo por ejemplo de la extensión del dominio, siendo los .com los más cotizados y otros como .net más asequibles.

Los dominios se adquieren a cambio de dinero a través de un registrador y luego pueden ser transferidos. Antes de adquirir un dominio se ha de verificar que está disponible. La adquisición de estos dominios tiene una duración acordada con el registrador. Después, pueden ser renovados. El dominio prismasound.es ha costado 6€ y tiene la validez de un año. Si se quiere seguir usando habrá de ser renovado.

5.4 Limitaciones

Una de las mayores limitaciones que se encuentran al usar un servidor virtual privado son las bajas prestaciones que ofrecen. En este caso, el servidor solo contaba con 1GB de memoria RAM y 25 GB de disco duro. La mayor limitante en este tipo de aplicación es la memoria RAM.

El proceso de ejecución de la red requiere una carga computacional algo elevada y consume una cierta cantidad de RAM. Conforme aumenta la complejidad de este proceso (se aumenta la cantidad de fuentes a separar, canciones más largas, varias peticiones…) la memoria se satura y los procesos son destruidos. Esto supone una degradación seria del servicio ya que impide su funcionamiento y es una de los factores más limitantes.

Una solución podría ser contratar un servidor dedicado que contase con unas prestaciones mayores. Esto solucionaría este problema prácticamente por completo.

Otra solución parcial sería crear un ‘swap file’. Se trata de reservar un espacio de memoria del disco duro para que la RAM almacene temporalmente información ahí cuando esta se llena. En esta aplicación se ha optado por esta solución que es económicamente más viable que un servidor

dedicado que puede incrementar enormemente el precio. Supone sacrificar espacio en el disco duro a cambio de permitir a la memoria RAM tener algo más de capacidad. Sin embargo, cabe destacar que esta solución también tiene sus limitaciones en cuanto al rendimiento.

Una limitación algo menor puede ser conseguir un dominio web atractivo para atraer el máximo de visitas a nuestro servicio. Muchas veces, los dominios ya están reservados o tienen unos precios muy altos, habiendo compañías que han llegado a pagar millones de dólares para comprar uno de estos preciados dominios.

Capítulo 6. Conclusiones

6.1 Resultados

Podemos ver como la situación de la separación de audio va viento en popa gracias al Deep Learning, por el cual multitud de diferentes organizaciones están trabajando con distintos métodos para ir consiguiendo cada vez una solución mejor ante este problema.

A priori, se puede decir que es posible servir una red neuronal mediante Node.js de forma satisfactoria. Sin embargo, podemos ver que es una tarea que consume cierta cantidad de tiempo. Este tiempo depende de la duración del archivo de audio entre otras cosas. Esto puede ser problemático debido a la naturaleza de funcionamiento de Node. Si se juntan varias peticiones muy caras puede llegar a un estado en el que se bloqueen todos los hilos de ejecución I/O y se produzca un cuello de botella fatal para el servidor.

No obstante, una carga baja de trabajo puede mantenerse. Además, todo indica a que, si se hubiera conseguido servir el modelo de Tensorflow de forma nativa, el rendimiento aumentaría considerablemente ya que se ejecutaría todo sobre JavaScript y no habría que comunicarse con subprocesos ni hacer tareas en segundo plano. Además, los modelos solo serían cargados una vez cuando se iniciase la aplicación, reduciendo enormemente el tiempo de procesado de todas las peticiones.

Pese a estas dificultades, se puede considerar una buena alternativa utilizar Node.js para servir redes neuronales mediante web siempre y cuando estas redes sean lo más ligeras posibles para el servidor o se vayan a ejecutar en el lado del cliente. El desarrollo ha sido cómodo y el resultado robusto y fácilmente escalable, además de contar con una buena perspectiva de futuro.

Sería interesante estudiar también el desempeño de este tipo de implementaciones delegando la ejecución al navegador del cliente mediante Tensorflow.js ya que liberaría por completo la carga computacional pesada del servidor y sería una situación ideal para utilizar Node.

También se ha demostrado cómo es posible desplegar un servicio de estas características en un servidor privado virtual sin demasiadas complicaciones. Sin embargo, ha quedado expuesto que para que este tenga un buen funcionamiento, va a ser preciso que se contrate un servidor dedicado para que el servicio tenga unas buenas prestaciones. Un servidor de bajas prestaciones puede suponer el no funcionamiento del servicio y debe ser tenido en cuenta en la planificación inicial.

6.2 Desarrollo futuro

Como desarrollo futuro se ha pensado que algunas funciones requieran registrarse para poder ser usadas. Por ejemplo, que sin registrar solo se pueda usar la separación de dos fuentes y en mp3. Por lo tanto, al iniciar sesión se desbloquearía el resto de configuraciones adicionales.

Añadir una escala para puntuar lo bien que se han separado las pistas de cara a una futura mejora de la red. De esta manera, se obtendría información sobre el género musical con el que mejor funciona la aplicación.

Por otra parte, se podrían añadir nuevos recursos REST para realizar más funciones de procesado del audio o interacción con los datos, como obtener estadísticas del número de archivos separados por cada usuario, mezclar pistas, etc… También se podría añadir una pasarela de pago para cobrar por el acceso a la API del servicio web y así monetizarlo para que pudiera cubrir los gastos generados por el mantenimiento.

Otro hilo por el que se puede seguir podría ser convertir los modelos de Tensorflow a Tensorflow.js para probar una implementación de la red en el lado del cliente. Esto supondría un abanico de nuevas posibilidades a la hora de desarrollar redes de baja carga computacional.

Una vez añadidas todas estas mejoras, se podría plantear una versión *standalone* de la aplicación para poder ser usada como aplicación de escritorio. Existen frameworks como Electron.js que

permiten construir aplicaciones de escritorio desarrolladas con JavaScript, por lo que no supondría apenas dificultad añadida, ya que la mayoría del código sería reutilizable.

Finalmente, sería interesante que se solventase el problema por el cual no se ha podido utilizar el modelo nativo de la red mediante Node.js, lo cual podría incrementar en gran medida el rendimiento de la aplicación.

Capítulo 7. Bibliografía

Al-Zoubi, K., & Wainer, G. (2009). Using REST Web-Services Architecture for Distributed Simulation. *ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*. Obtenido de https://ieeexplore.ieee.org/abstract/document/5158326

Ann Yuan. (06 de 2020). *World Wide Web Consortium (W3C)*. Obtenido de https://www.w3.org/2020/06/machine-learning-workshop/talks/fast_client_side_ml_with_tensorflow_js.html

Barrios, J. (2019). *Juan Barrios*. Obtenido de https://www.juanbarrios.com/redes-neurales-convolucionales/

Défossez, A., Usunier, N., Bottou, L., & Bach, F. (2021). Music Source Separation in the Waveform Domain. *HAL*. Obtenido de https://hal.archives-ouvertes.fr/hal-02379796/document

Dialani, P. (25 de 11 de 2020). *Analytics Insight*. Obtenido de https://www.analyticsinsight.net/top-6-machine-learning-trends-of-2021/

Dobrev, D. (2005). Dobrev D. A Definition of Artificial Intelligence. *Mathematica Balkanica*.

El Naqa, I., & Murphy, M. (s.f.). What Is Machine Learning? *Machine Learning in Radiation Oncology. Springer, Cham*.

GeeksforGeeks. (13 de 03 de 2020). *GeeksforGeeks*. Obtenido de https://www.geeksforgeeks.org/node-js-event-loop/

Kehtarnavaz, N. (2008). *Digital Signal Processing System Design.*

Microsoft. (21 de 04 de 2021). *Microsoft Docs*. Obtenido de https://docs.microsoft.com/es-es/azure/machine-learning/concept-deep-learning-vs-machine-learning

Robjohns, H. (03 de 2010). *Sound on Sound*. Obtenido de https://www.soundonsound.com/sound-advice/buying-used-tape-machine

Roinneberger, O., Fischer, P., & Brox, T. (2015). *Medical Image Computing and Computer-Assisted Intervention.* Springer.

Zhang, W., Yang, G., Lin, Y., Ji, C., & Gupta, M. (2018). On Definition of Deep Learning. *World Automation Congress (WAC)*.